Graduate Atmospheric Dynamics Laboratory 1:
Global changes in lower-tropospheric conditions over the recent climatological period

Stefan Rahimi
Tuesday 30 August 2022

Due date: Tuesday 6 September by the start of lab
Deliverables: You will turn in a Jupyter notebook containing:

1. A properly working data reader function for ERA5 with the requested functionalities (below)
2. A properly working spatial plotting function with the requested functionalities (below) along with a subsequent figure of the spatiality lower tropospheric temperature across the globe
3. Figures requested (temperature trend during the recent climatology)

You will be graded on figure presentation quality as well as functionality, so please spend time to create presentable figures!

Turning your labs in
Upon completion, you will copy your lab into a restricted directory on GLADE named /glade/work /<username>/lab4dynamics (this will be linked to your /home directory). Make sure this is a **restricted directory** by assigning the appropriate permissions to the directory. You don't want others to copy your work!

Synopsis
Throughout the lab sequence, you will perform geospatial analyses on NCAR-Wyoming Supercomputing (NWSC) resources to develop an intuitive understanding of the fundamentals of atmospheric dynamics. Lab 1 introduces you to **JupyterHub**, the python construct in which you will be developing and implementing your analysis software.

By now, it is assumed that you have full access to NWSC systems.

1. Setting up your conda environment

As explained previously, it is considered best practice and advantageous to have multiple library configurations at your fingertips. Python is a highly compartmentalized code base developed by countless users and is constantly being updated. If you spend months perfecting a code to function within NCAR's Python library (NPL), your code could break when NCAR updates NPL, forcing you to modify the code. Hence, the first item of business in the lab is creating a static and fully customizable conda environment on NWSC systems.

1.1. Inside a terminal window, ssh onto a Casper login node (ssh -Y <username>@casper.ucar.edu)
1.2. 'module load conda' – conda must be loaded!
1.3. 'module save' – saves your modules so you don't have to load conda every time you log in….
1.4. 'module list' – lists your active, inactive, and hidden modules
1.5. 'cnda env list' – list the available conda environments
1.6. NPL is a great environment, but we cannot customize it. You want to create a clone of NPL that you can activate, use, and customize. To clone the NPL environment to a new environment, type: 'conda create --name dynamics_class --clone npl. This command creates a new environment called dynamics_class which is a clone of npl.
1.7. Activate your new environment by typing 'conda activate dynamics_class'
1.8. Your new environment will not automatically be accessible on NCAR's JupyterHub portal. To make it so, type the following on your command line: 'python -m ipykernel install --user --name=dynamics_class
1.9. Now in a web browser, navigate to https://jupyterhub.hpc.ucar.edu/, click 'production', authenticate, and log in.
1.10. Under 'Service Name', name your instance and click 'Add New Server'
1.11. Select Casper Login, and JupyterHub will initialize with your home directory contents visible
1.12. On the right pane, you will see the kernel 'dynamics_class' as a potential library to use under 'Notebook'.
1.13. Back in your terminal window, navigate to your /work directory
1.14. Create a directory called labs4dynamics
1.15. Symbolically link labs4dynamics to your /home directory
1.16. Back in your Jupyter browser window, you should see your newly created directory labs4dynamics. Navigate to it. You will want to save all of your labs to your work directory because you have more storage space there! At any time, you may chech your storage limits by typing 'gladequota'.
1.17. Create a notebook by clicking on dynamics_class
1.18. In your first cell, insert the following lines:
```
import sys
print(sys.prefix) #shows where the interpreter installation is located
```
1.19. Execute the cell. You can also do this by typing Shfit+Enter
1.20. Returned is a path to where your dynamics_class environment is located, thus confirming your successful cloning, activation, and usage in Jupyter.

2. **Lab assignment**

Today, we are working with reanalysis data from the European Centre for Medium-Range Weather Forecasts (ECMWF), specifically their 5th atmospheric reanalysis, ERA5. A reanalysis dataset reconstructs historical weather and climate by combining the results of numerical simulations, governed by the laws of physics, with observations via a process called data assimilation. Because the outputs of ERA5 are (at least mostly) physics-based, we have access to variables that are difficult to observe.

The first thing to do when working with datasets is to create a data loader. Firstly, we must know where the data are located and secondly how they are organized. This exercise is performed every time we work with a new and unfamiliar data set.

ERA5 monthly mean data on atmospheric pressure levels are located on GLADE in:
/glade/collections/rda/data/ds633.1/e5.moda.an.pl

These data are subset by year, so,

1980 data are in: /glade/collections/rda/data/ds633.1/e5.moda.an.pl/1980
1981 data are in: /glade/collections/rda/data/ds633.1/e5.moda.an.pl/1981
1982 data are in: /glade/collections/rda/data/ds633.1/e5.moda.an.pl/1982
etc.

You may print out the file metadata, coordinate, and variable information by using **ncdump**. Specifically,

ndump -h e5.moda.an.pl.128_131_u.ll025uv.1979010100_1979120100.nc

will print out the file's contents to the command line.

Personally, I prefer to use **ncview** for quick and easy visualization using .nc (NetCDF4 files). In some ways it is more user friendly and provides more flexibility, especially when exploring 3- and 4-dimensional data or model output. Instead of being provided a full list of the variables, you are typically given drop-down lists for the variables separated based on the number of dimensions (if you have only a few variables, they will be listed). After your module load ncview, you may type:

ncview e5.moda.an.pl.128_131_u.ll025uv.1979010100_1979120100.nc

The terminal window will interface with your personal computer to open a separate window. This happens differently depending on whether you are using a Mac or Winnows machine. Regardless of the machine used, I highly recommend getting this functionality to work for your own research benefit!

2.1. **Data Reader**

Create a data loader function called _DataReader that reads in multiple .nc data files from ERA5, concatenating them along the time dimension, and organizes them into a unified dictionary. The variables that should be included are 3-dimensional temperature, the u-component of the wind, the v-component of the wind, and specific humidity. This data reader should have these capabilities:

a. Specify the time interval of the read. Specifically, give your data reader the functionality to only work with, for example, years 1990-2000, while neglecting other years from the reanalysis.
b. Specify the latitude and longitude bands

c. Load the data numerics into memory

Hints: This function should make use **xarray.open_mfdataset(), glob.glob**, **xarray.sel()** along with the **slice()** function, and **xarray.load()**. Xarray is a python library that doesn't actually load numerical data into memory unless directed to. Even after a file is opened, non-coordinate arrays are not actually loaded into resident memory but rather placeholders to the actual data are stored 'under the hood'. This allows users to see the array attributes, run code correctness checks, etc., until they are ready to unleash their calculations on the full arrays. To visualize your latitude and longitude delimiters, try selecting (.sel or .isel) at a time element and plot your array using **array.plot()** (for .plot() to work, 'array' must be an xarray).

I developed my data reader by creating an array of the files with absolute paths to the files that I wanted to open for each individual variable before using xarray.open_mfdataset(). I then iterated (looped) through my desired variables, assigning my loaded data to the dictionary. Upon using xarray.open_mfdataset, I set **combine='by_coords'** to concatenate data along the time axis.

The result here will be an abstraction in which _DataReader will be called 4 times (once for each variable), and the dictionary content will be assigned iteratively.

Using _DataReader, create a dictionary, reading in data from 1980-2014. Use data from the whole globe. We will subset the globe in subsequent labs.

My data reader call looks like:

```
_DataReader(variable,nc_VariableName,year1,year2,dir,
                lonMinimum,lonMaximum,
                latMinimum,latMaximum)

#variable = variable naming convention in file name
#nc_VariableName = variable name within the .nc file
#year1 = start year of files considered
#year2 = end year of files considered
#dir = where are the pl ERA5 files located
#lonMinimum = lower-left longitude bound to consider
#lonMaximum = upper-right longitude bound to consider
#latMinimum = lower-left latitude bound to consider
#latMaximum = upper-right latitude bound to consider
```

2.2. First plot

First, let's make a function for a simple plot which displays the spatial distribution of a quantity, say temperature.

Let's begin by plotting 925 hPa temperature. To begin, we define our figure. We will be conducting our geospatial analyses in matplotlib and **cartopy**, so we must import the requisite libraries:

```
import matplotlib.pyplot as P
import cartopy.crs as ccrs
import cartopy.mpl.ticker as cticker
```

```
from cartopy.util import add_cyclic_point
```

Now define the **figure**:

```
fig = P.figure(fignum,figsize=(11,8.5),dpi=300)
fig.patch.set_facecolor('xkcd:white') #Set the background = 'white'
```

Next up, we must define the **projection** using cartopy:

```
ax = P.axes(projection=ccrs.PlateCarree()) #You may use whatever projection you like
```

Next, we must deal with the fact that our **longitudinal** axis is **periodic**

```
data2plot, lons2use = add_cyclic_point(<netcdf data>, coord=<longitude array>)
```

If you don't insert the above line, there will be an awkward swatch of missing values along the prime meridian.

Next, we plot using:

```
    cs = ax.contourf(lons, latitude, data2plot,
            transform = ccrs.PlateCarree(),
            cmap=color_map,extend='both',
            levels=levels)

#levels = contour intervals for your plotting; I define mine below
```

And add: **coastlines**, **gridlines**, a **figure title**, and a **colorbar**.

Also, try adding:

```
P.tight_layout()
```

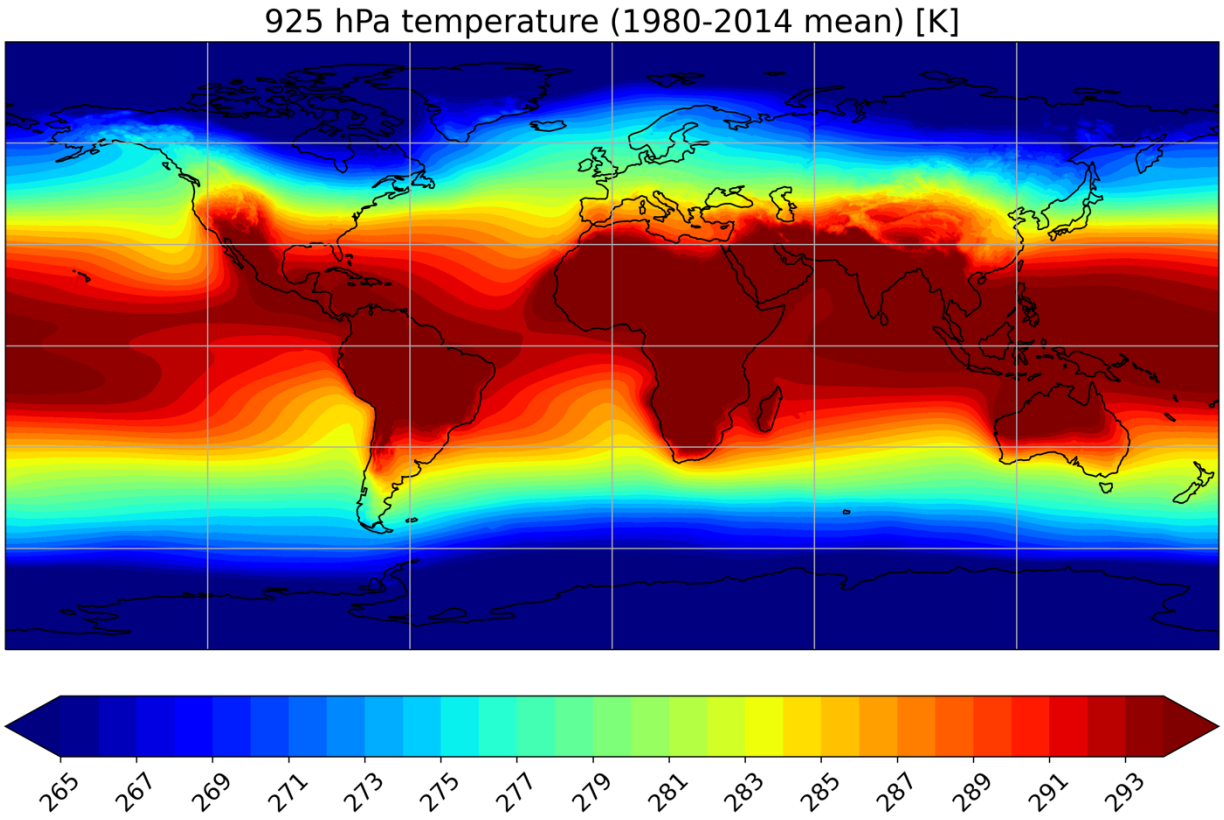
within your plotting function to see if it improves the display. Try playing around with the **pad** keyword in your colorbar definition to more appropriately position it.

Save the resulting figure by typing:

```
P.savefig('./%s.png' %(filename),dpi=300)
```

To confirm that this function is working, **plot the 1980-2014 mean temperature** as a function of latitude and longitude.

My figure is:



My figure maker call looks like:

```
_FigureMaker(fignum,array,variable_name,
                levels,color_map,filename)

#fignum = figure number within the cell. If figures are in different cells, then
#fignum = 1 always
#array = array to plot. Must have dimensions f(lat,lon)
#variable_name = title of figure
#levels = range of values over which to contour, including intervals. I set mine as
#levels = np.arange(min,max,interval)
#color_map = P.cm.<colormap name from matplotlib.pyplot>
#filename = save name for output figure
```

2.3 Recently observed climate trends

We are going to compute the linear trends in our 4 variables at each grid points using a simple linear regression:

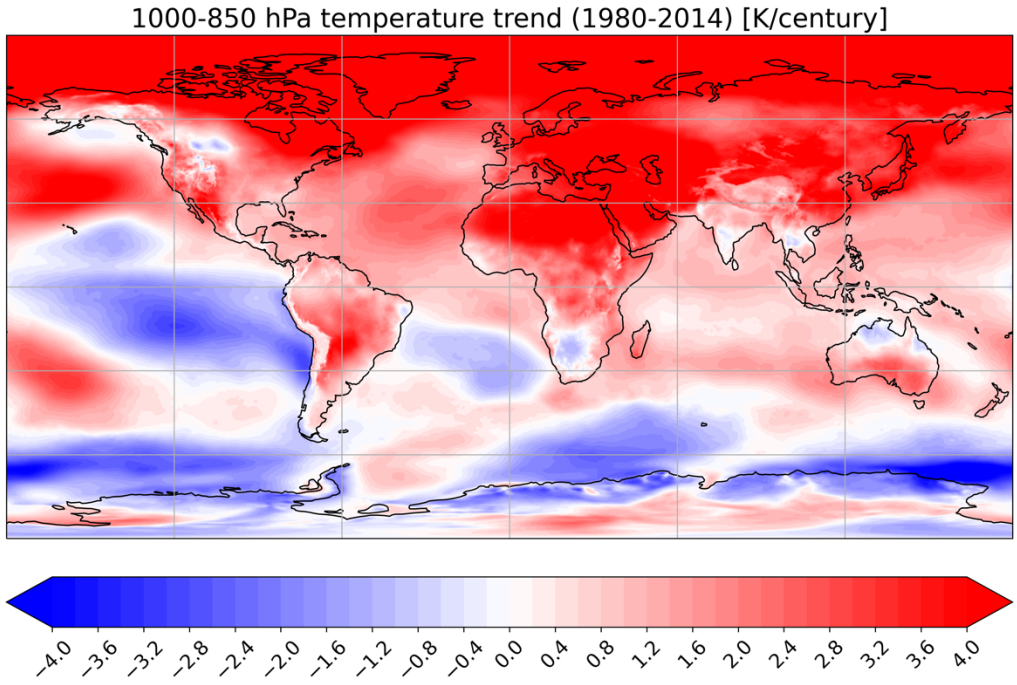
```
def _linregress(data):

    import numpy as np
    from scipy import stats

    #Input array is 1-D in time

    #This function computes the linear trend in a time series
```

```
    #using stats.linregress. The final units are per time step
    #of data. E.g., if the data are monthly means, then the
    #Returned trend is in <units>/month

    #Function computes linear regression at individual gridpoint

    slope = stats.linregress(np.arange(0,len(data)),data).slope
    return (slope)

def _trends(array):

    #Input array is f(time,latitude,longitude)

    #Applies _linregress at each spatial grid point

    trend = xr.apply_ufunc(
        _linregress,
        array,
        input_core_dims=[["time"]],  # list with one entry per arg
        exclude_dims=set((["time"])),  # dimensions allowed to change size. Must be a set!
        vectorize=True,  # loop over non-core dims
        )

    return ( trend )
```

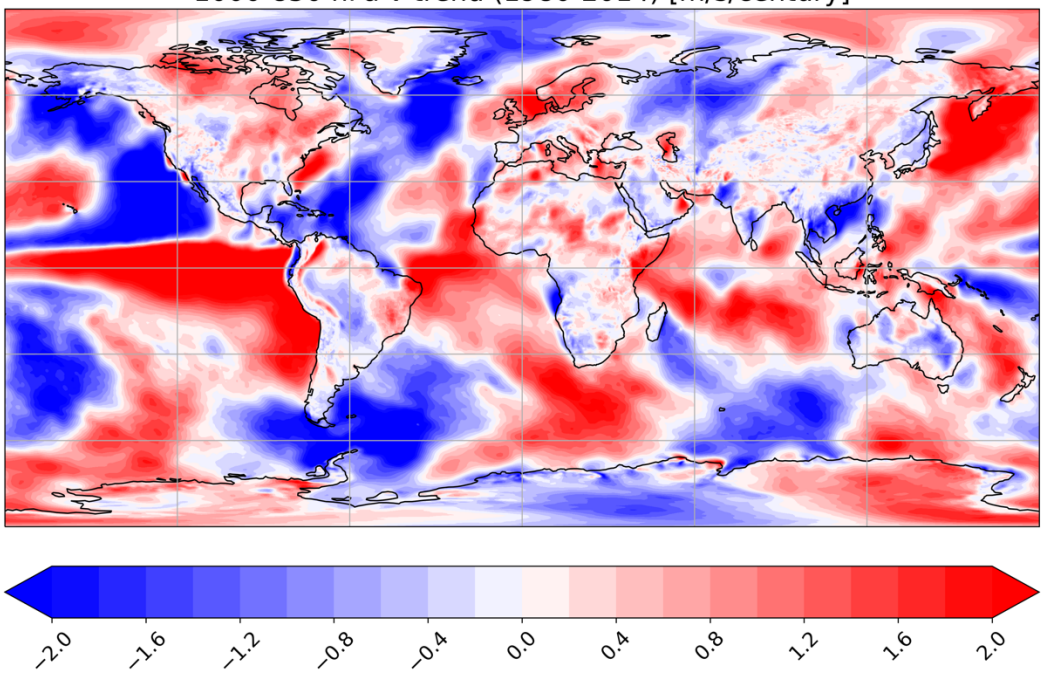
Using the above functions, plot linear trends in:

1. 1000-850 hPa mean temperature
2. 1000-850 hPa mean u
3. 1000-850 hPa mean v
4. 1000-850 hPa mean specific humidity

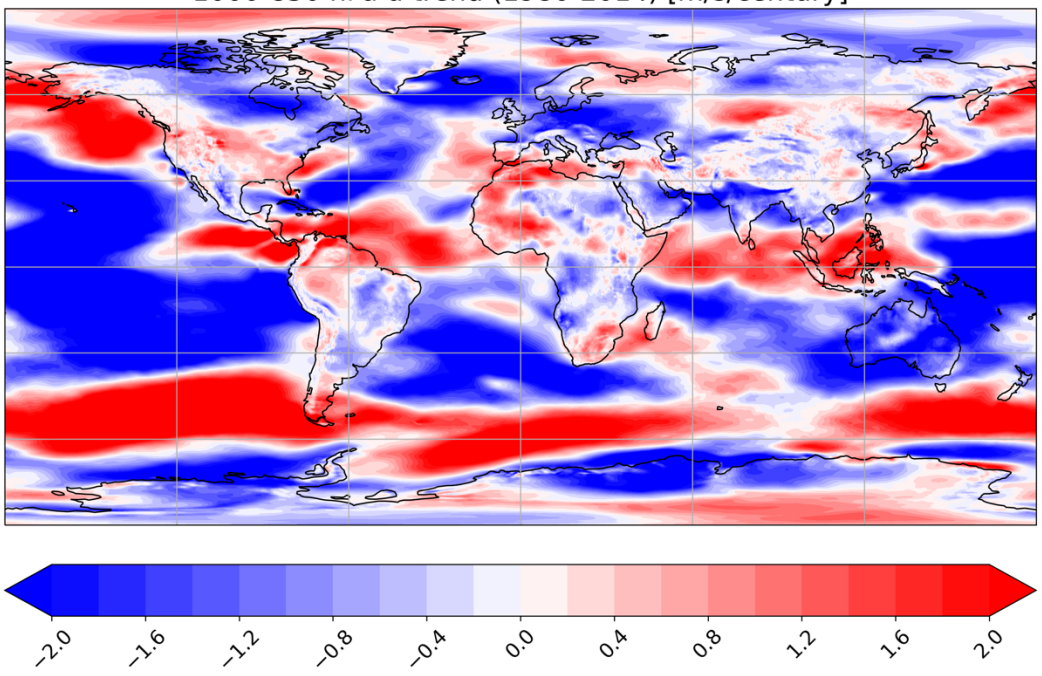
My figures look like:

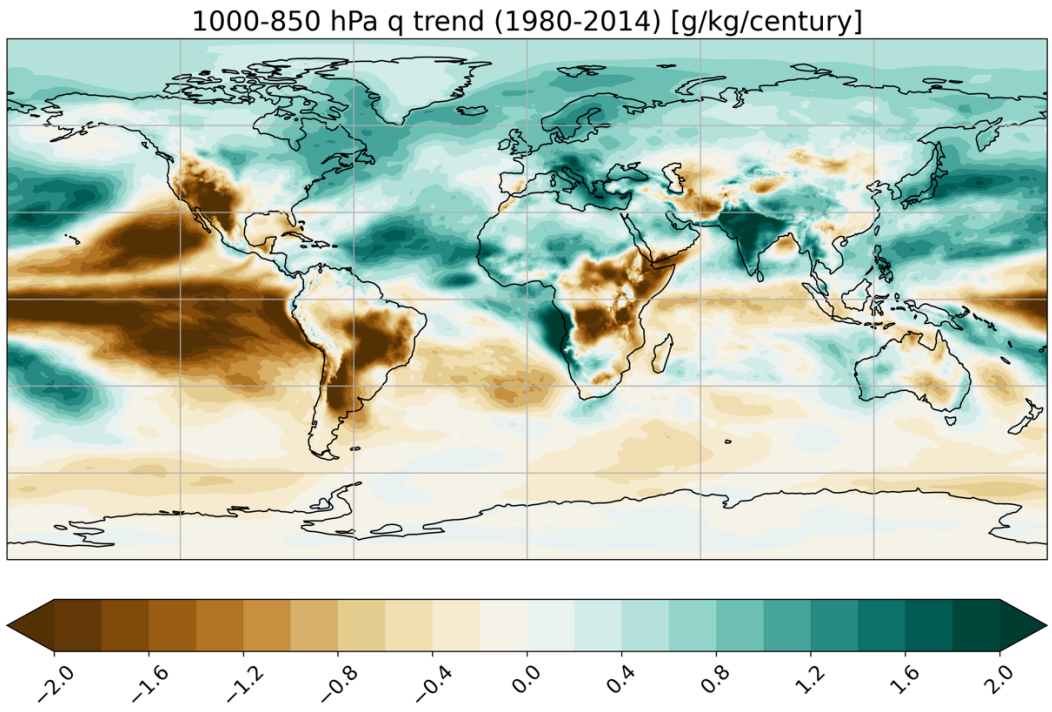


1000-850 hPa v trend (1980-2014) [m/s/century]



1000-850 hPa u trend (1980-2014) [m/s/century]





Next week, we will lead off lab by linking the changes in winds to changes in mean sea-level and lower-tropospheric pressure.